High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance

Andrew Chien Scott Pakin Mario Lauria Matt Buchanan
Kay Hane Louis Giannini
Jane Prusakova
Department of Computer Science and
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
{*achien,pakin,lauria,mbbuchan,hane,giannini,prusakov*}*@cs.uiuc.edu*

December 21, 1996

Abstract

The HPVM project provides software which enables high-performance computing on clusters of PCs and workstations using standard supercomputing APIs such as MPI, SHMEM Put/Get, and Global Arrays. HPVMs—High-Performance Virtual Machines—are surprisingly competitive with MPP systems, such as the IBM SP2 and Cray T3D. The Illinois HPVM achieves impressive low-level communication performance across the cluster: one-way latencies of around 11 μsec and bandwidths $>$ 50 MBytes/sec—even for small packets ($<$ 256 bytes). Performance at higher levels, such as MPI, is expected to be approximately 17 μsec latency and also $>$ 50 MByte/sec bandwidth.

Current elements of the HPVM project are: Illinois Fast Messages (FM), MPI-FM, FM-DCS (Dynamic Coscheduling), Put/Get-FM, and Global Arrays-FM. This software is in use at dozens of sites.

1 Introduction

The advent of high-performance microprocessors and their integration into low cost computing systems is causing a fundamental realignment in the way that high-performance computing is used and delivered. As microprocessors have continued to increase in performance, they have approached the performance of the fastest vector processors and, because of their low cost, have become the building block of choice for high-performance parallel computing systems. Machines such as the Intel Paragon [12], Thinking Machines CM-5 [24], IBM SP2 [11], and, more recently, SGI/Cray's T3D [5], T3E [19], and Origin 2000 [20] are all based on microprocessors, and combine them in parallel configurations to achieve high performance. This unification of processor designs between high-performance computers and the desktop yield interesting synergies: high-end systems can benefit from low-end software, and high-end systems can leverage high-volume subassemblies such as processors, system boards, or even entire systems. The primary remaining distinguishing feature of high-performance systems from distributed systems is the speed of communication and efficiency of coordination.

While parallel machines typically exploit high-performance custom networks, high-performance commodity networks are becoming widely available at reasonable cost [2, 3, 7, 10]. While it may be some time before these networks are pervasive, their ready availability makes building machine clusters with high-performance interconnects feasible. For example, Myricom's Myrinet [2] hardware is capable of link speeds of 160 MBytes/sec and switch latencies of under a microsecond. The advent of such high-performance networks coupled with gigaflop microprocessors makes high-performance

computing on clusters an attractive alternative. One of our goals in the High Performance Virtual Machines (HPVM) project is to deliver high-performance computing from distributed computational and network resources.

Based on an opportunity which arises from the availability of high speed interconnects and recent advances in communication technology, we are trying to unify the two major cluster models: resource stealing (to increase job throughput) and dedicated cluster (to achieve high aggregate performance). The resource stealing model, pioneered by systems such as Condor [14] and Utopia [28] and now commercialized in systems such as the Load Sharing Facility [23], exploit heterogeneous, shared resources connected by low-performance networks and harness them to achieve high throughput of sequential jobs. If parallel jobs are supported, they are efficient only if they are loosely-coupled (e.g. as in PVM). The dedicated cluster model (e.g. the IBM SP2 or Berkeley NOW) exploit homogeneous, dedicated resources connected by a high speed interconnect to achieve high performance. Efficient coordination can be achieved because resources are dedicated and uniform.

The objective of the HPVM project is to unify these models to deliver high-performance parallel computing on shared, heterogeneous resources. The critical challenges include:

- delivering high-performance communication to standard, high-level APIs,
- coordinating scheduling and resource management, and
- managing heterogeneity

Delivering high performance presumes the availability of processors, memories, and interconnects with appropriate hardware performance characteristics. In this paper, we address the delivery of high performance communication. For a discussion of coordinated scheduling—another key aspect of HPVM—see [21].

Communication performance is a critical aspect of parallel computation, so we have built a series of low-level and high level communication layers which deliver high-performance communication for distributed resources. The lowest layer, Illinois Fast Messages (FM) achieves 11 μsec latencies and bandwidths in excess of 50 MBytes/sec. Atop this layer, we have built a range of high level interfaces, including the Message Passing Interface (MPI), SHMEM Put/Get, and Global Arrays. Indications from previous versions of FM are that we can deliver the full bandwidth of FM to these higher level communication layers with only a modest increase in latency.

Together, high-performance communication and coordinated scheduling provide the basis for high-performance parallel computing on shared, distributed resources. As an initial demonstration this technology, we ran Zeus-MP, a hydrodynamics code used for cosmology, atop the MPI interface of HPVM. This code achieves linear speedups and good absolute performance on a cluster of uniprocessor Pentium Pro Machines. Further performance numbers both for additional high level APIs and applications will be presented in the talk.

The remainder of the paper is organized as follows. Section 2 summarizes the technological trends which motivate this work, as well as related work on high-performance communication and coscheduling. In Section 3, we briefly summarize the design and performance of FM and the high level communication APIs supported by HPVMs. Section 4 pulls this all together, and describes performance on the Zeus-MP code. Finally, in Section 5, we briefly summarize the results.

2 Background

An important technology trend is the rapid increase in microprocessors' computing performance. Low-cost microprocessors are arguably as powerful today as any computer processors that can be built. The unavoidable market implication of low-cost microprocessors with high absolute performance is that high-performance systems must be constructed from scalable ensembles of microprocessors. Virtually all high-performance computing vendors now market systems based on ensembles of microprocessors.

In addition to fast microprocessors, fast networks are also a necessity for efficient coordinated computation. Today, most local area networks are interconnected via 10 Mbits/sec Ethernet [17]. However, 10 Mbits/sec is far too little bandwidth for network-intensive applications. Fortunately, paralleling the advances in microprocessor performance, a number of new, higher-bandwidth

"killer networks" have recently hit the market. These include FDDI [6], 100 Base-T Ethernet [7], FibreChannel [27], Myrinet [2], and ATM/SONET [3] and currently run from 100 Mbits/sec to over 600 Mbits/sec, with the ability to scale to Gbit/sec bandwidths and beyond.

Unfortunately, software for these new networks generally lags well behind the hardware in terms of performance. Legacy software structures, designed more for portability/interoperability than for performance, are still prevalent. The key problems are operating system involvement and redundant memory copies, both of which add substantial overhead to message processing that can be tolerated on slow networks, but not on more recent, faster networks. There are a large number of high-performance communication projects that address these problems by removing the operating system from the critical path of communication. Instead, the operating system is used solely for initializing the network interface, establishing connections with other nodes, mapping network interface control registers and memory buffers into user space, and tearing down connections when communication is complete. Data transmission—the common case—is performed at user level. This approach is exemplified by Hamlyn [4], Cranium [15], U-Net [25], and SHRIMP [1]. Since operating systems are traditionally used for process protection, these systems all exploit the system's virtual memory hardware for protection, a much lower-overhead mechanism than system calls.

With fast microprocessors, high-speed networks, and efficient software, PCs are an effective tool for high-performance computing. In fact, PCs are already an established platform for scientific computing. PCs are the primary vehicle for CAD tools, numerical software packages such as MATLAB, and other mathematical/scientific programs that are mainstream for computational scientists. Nevertheless, HPVM greatly extends the domain of scientific computing on commodity systems by enabling sets of computers equipped with high-speed networks to solve problems previously possible only on MPPs and supercomputers.

3 High-performance Communication

While hardware technology has provided high-performance communication's building blocks—high-speed links, fast routers, and low-cost network interface cards—traditional software and hardware system structures make it hard to deliver the hardware performance to applications. In fact, the predominance of software in introducing delay and overhead has come to be called the "last inch" problem, a reference to the "last mile" problem faced by telecommunications corporations where most of the overhead is incurred in the last mile of distance. Traditional networking software (e.g. TCP/IP) is designed for 10 Mbit/sec Ethernet and incurs overheads in the range of 100 μsec per packet. In contrast, currently-available high-speed networks such as Myrinet are two orders of magnitude faster.

Because our goal is to match or surpass communication performance in integrated parallel machines, radical changes in software architecture are necessary. To exploit the full performance of these high-speed interconnects, we have designed Illinois Fast Messages (FM), a low-level portable communication interface which efficiently delivers the hardware performance to higher levels. In the following subsections, we describe first the FM interfaces, and then several of the higher level communication layers built on top.

3.1 Fast Messages

The FM interface traces its roots to Berkeley Active Messages [26] on the CM-5. The FM 1.1 API (Table 1) contains functions for sending long and short messages and for extracting messages from the network. Each message is associated with a *handler*, a function executing at the receiver and that stores or processes the message data. What distinguishes FM from other messaging layers is not the surface API, but the underlying semantics—the service guarantees and control of the memory hierarchy that FM provides to software built atop FM. Analysis of the literature and our ongoing studies to support fine-grained parallel computing [13, 22] led to the conclusion that a low-level messaging layer should provide the following key guarantees, or higher-level messaging layers will suffer a performance loss:

- Reliable delivery,
- Ordered delivery, and

- Control over scheduling of communication work (decoupling).

TABLE 1
FM 1.1 API

Function	Operation
`FM_send(`*dest,handler,buf,size*`)`	Send a long message
`FM_send_4(`*dest,handler,i0,i1,i2,i3*`)`	Send a four word message
`FM_extract()`	Process received messages

We initially implemented the FM 1.1 interface on the Cray T3D and a cluster of SPARCstations connected by Myrinet. The fact that we were able to port FM to two radically different systems—a heavily-integrated MPP and a workstation cluster—is a testament to the portability of the interface. Furthermore, FM 1.1's latency and bandwidth were not just excellent in absolute terms, but also substantially better than the vendor-supplied messaging layers on each system (Table 2).

TABLE 2
FM performance

Performance metric	Cray		Cluster		
	PVM	FM 1.1	Myrinet API	FM 1.1	FM 2.0
Latency (μsecs)	28.0	6.1	102.1	12.0	13.7
Bandwidth (MBytes/sec)	37.5	112.9	4.5	16.1	17.1

Even though FM 1.1 performed well on the latency and bandwidth microbenchmarks, saved higher-level messaging layers from having to implement reliable, ordered delivery, and allowed control over communication scheduling, we still observed that much performance was lost when higher-level messaging layers were implemented on top of FM. We identified three problems:

1. the fact that we allowed control over *when* communication data is processed, but not *how much* data is processed
2. the cost of copying a user-specified buffer merely to prepend a header to it before passing the data to FM
3. the cost of copying data from FM's internal buffer into a messaging layer buffer and then into the user-specified buffer.

To address these problems, we redesigned the FM interface to support *data pacing* and a novel concept called *streamed messages*. Data pacing enables the programmer to limit the amount of data extracted by `FM_extract()`. And streamed messages are messages that can be sent and extracted piecewise. Streamed messages not only increase communication pipelining—the receiver can process a message even before the sender has finished sending it—but also reduce data copying. Using streamed messages, a messaging layer build atop FM can receive a piece of a message and, using information in that piece, decide where subsequent pieces of the message should be received to. Table 3 shows the FM 2.0 interface. While FM 2.0 performs as well as FM 1.1 on microbenchmarks, it delivers substantially more of the hardware performance to applications—one of the key goals of FM.

We implemented the FM 2.0 on SPARCstations and, more recently, on a Myrinet cluster of Pentium Pro-based PCs running Windows NT. What makes the PCs interesting from a high-performance communication standpoint is that the PCI bridge supports write combining, meaning that back-to-back writes to sequential addresses are aggregated and sent across the bus in a high-bandwidth burst transaction. Most other systems perform burst transactions only for DMA operations. Consequently, FM 2.0's bandwidth on the PCI-based PCs approaches the full PCI bandwidth. FM's peak bandwidth is also much greater than on the SBus-equipped SPARCstations—52.6 MBytes/sec versus 17.1 MBytes/sec. In addition, by avoiding the overhead of DMA setup, FM

TABLE 3
FM 2.0 API

Function	Operation
`FM_begin_message(`*dest,size,handler*`)`	Open a streamed message
`FM_send_piece(`*stream,buf,size*`)`	Add data to a streamed message
`FM_end_message(`*stream*`)`	Close a streamed message
`FM_extract(`*maxbytes*`)`	Process up to *maxbytes* of received messages
`FM_receive(`*buf,stream,size*`)`	Receive data from a streamed message into a buffer

achieves its bandwidth half-power point with message sizes of only 256 bytes. This is an important result, because most network traffic is comprised of messages of about that size (see, for example, [9]).

3.2 Standard APIs

While low-level layers such as Fast Messages can deliver hardware communication performance, higher-level layers offer greater functionality, application portability, and ease of use. The problem is that high-level layers add overhead to communication and generally perform significantly worse than low-level messaging layers. FM, and specifically, FM 2.0, is designed to reduce the performance gap between low- and high-level messaging layers. To demonstrate this capability (and to create a high-performance implementation of a high-level API), we implemented two high-level APIs atop FM: MPI [16] and Global Arrays [18].

MPI-FM is based on the MPICH code base from Argonne and Mississippi State University [8]. MPICH is implemented as a two-level structure. The Abstract Device Interface (ADI) contains only a small number of functions (≈25), and the rest of MPI (≈125 functions) is built on top of that. To run MPI on FM, we needed only port MPICH's ADI to communicate with FM calls. MPI-FM, operational since October, 1995, exhibits comparable performance on a Myrinet-connected cluster of SPARCstation 20s to an IBM SP2 of the same vintage. Not only does MPI-FM deliver lower latency than that MPP, it also delivers higher bandwdith for messages below two kilobytes. Beyond two kilobytes, the architecture of the SP2's MPI implementation, which uses DMA based message injection, achieves higher bandwidth than the FM implementation, which uses programmed I/O. Because our PCI-based Pentium Pro systems eliminate much of the data copying bottleneck with write combining, we expect to deliver much higher bandwidth communication from MPI on that system. (The port is still in progress.)

Our second HPVM interface, Global Arrays (GA), is a portable programming model that uses Put/Get semantics for accessing and manipulating sections of large arrays in a shared address space. The portable GA interface was originally developed on the Cray T3D and is now in use on the Intel Paragon, IBM SP2, and several shared-memory machines. To port GA to FM, we first ported the lower-level SHMEM library from the T3D to FM. (The T3D version of GA uses the SHMEM primitives). SHMEM, like GA, uses a Put/Get interface, but its API closely matches the T3D's hardware features and is therefore somewhat challenging to convert to a software version and retain good performance. In the talk, we will present detailed performance measurements of GA-FM and SHMEM-FM.

Table 4 shows the status of all the HPVM interfaces we are currently developing. They fall into three main categories (although HPVM is, of course, not limited to these): message-passing, global name space, and cache-coherent. These interfaces will allow a variety of parallel programs to easily migrate to distributed resources. However, true portability implies preserving the performance model (resource abstraction) in order to port a software architecture without change.

4 Overall Performance

To demonstrate the capabilities provided by the current High Performance Virtual Machines, we describe the implementation and performance on a high-performance application. As an example, we use Zeus-MP, a hydrodynamics code produced by the Cosmology NSF Grand Challenge team.

TABLE 4
HPVM interfaces

Interface	Type	Status
Fast Messages	Message Passing	Available
Message Passing Interface	Message Passing	Available
Winsock 2	Message Passing	Under Development
SHMEM Put/Get	Global Name Space	Functional
Global Arrays	Global Name Space	Functional
CORBA	Global Name Space	Planned
Cached Object System	Shared Objects w/ caching	Planned
Shared Virtual Memory	Shared Address w/ caching	Planned

This program is used to model a wide variety of astrophysical phenomenon in three dimensions and for a variety of boundary conditions. This code does not vectorize well, and generally achieves good speedups on parallel platforms.

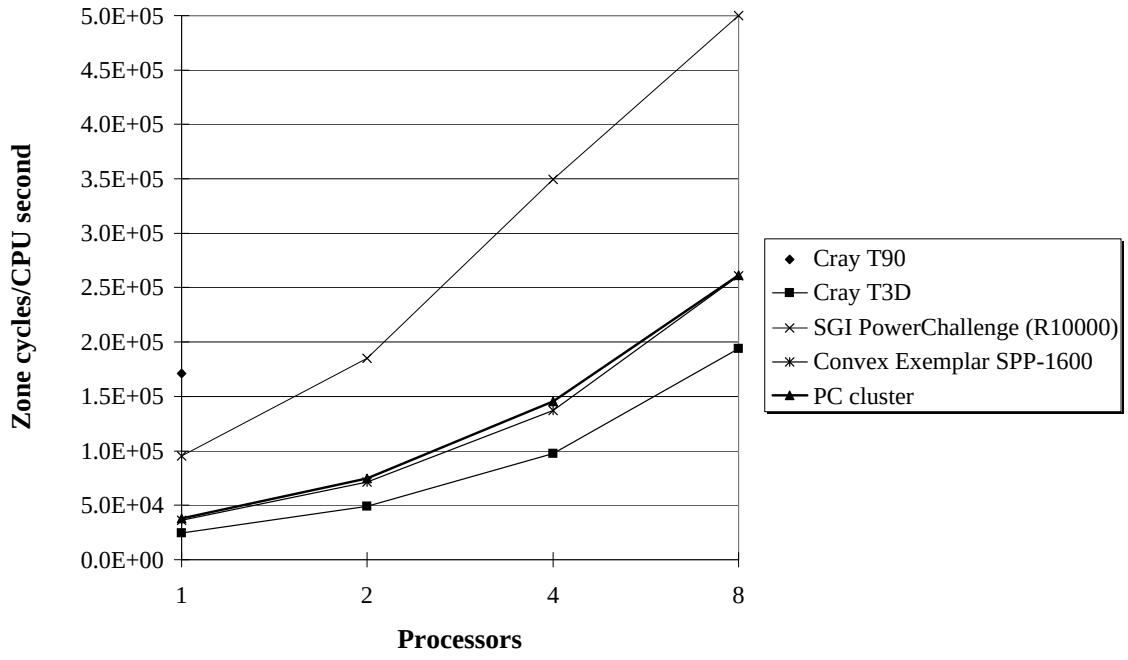

FIG. 1. *Scaled speedup of Zeus-MP on a variety of hardware platforms*

The Zeus MP code utilized the MPI application programming interface and was executed on our new testbed. This environment uses 200 Mhz Pentium Pro machines (256 KB L2 caches) and second generation Myrinet (LANai 4.1 boards and 160 MByte/sec links). The performance of distributed resources in our cluster (Figure 1) not only surpassed several recent parallel machines, it also exceeds performance on a Cray T90 processor with about 4–5 Pentium Pro nodes. To put things in perspective, comparison to a SGI Power Challenge, equipped with the latest R10000 processors, shows that Pentium Pros still trail the fastest microprocessors on floating point by a healthy margin.

5 Summary and Future Work

We have demonstrated MPP class communication performance through a number of communication interfaces on distributed computational and networking resources. These results raise interesting opportunities for greater exploitation of distributed resources and also wider availability of high-performance computing.

While we have made significant strides in making distributed resources attractive vehicles for high-performance computing, significant research challenges remain. We plan to experiment with larger clusters, including multiprocessor elements. At present, a 32-node cluster 2-way symmetric multiprocessor Pentium Pro machines is being deployed at Illinois. In addition, a number of questions remain about how to achieve effective coordination in a distributed environment. Finally, the dimensions of heterogeneity, adaptivity and fault tolerance, and the wide area are only beginning to be explored.

More information on the HPVM project is available from our WWW site, `http://www-csag.cs.uiuc.edu`.

Acknowledgements

The research described in this paper was supported in part by DARPA Order #E313 through the US Air Force Rome Laboratory Contract F30602-96-1-0286, NSF grants MIP-92-23732, ONR grants N00014-92-J-1961 and N00014-93-1-1086 and NASA grant NAG 1-613. Support from Intel Corporation, Tandem Computers, Hewlett-Packard, Microsoft, and Motorola is also gratefully acknowledged. Andrew Chien is supported in part by NSF Young Investigator Award CCR-94-57809.

In addition, this material is based upon work supported under a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg, *Virtual memory mapped network interface for the SHRIMP multicomputer*, in Proceeding of the International Symposium on Computer Architecture, April 1994, pp. 142–153. Available from `http://www.cs.princeton.edu/shrimp/papers/isca94.paper.ps`.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, *Myrinet—a gigabit-per-second local-area network*, IEEE Micro, 15 (1995), pp. 29–36. Available from `http://www.myri.com/research/publications/Hot.ps`.

[3] J. Boudec, *The Asynchronous Transfer Mode: A tutorial*, Computer Networks and ISDN Systems, 24 (1992), pp. 279–309.

[4] G. Buzzard, D. Jacobson, S. Marovich, and J. Wilkes, *Hamlyn: A high-performance network interface with sender-based memory management*, in Proceedings of the IEEE Hot Interconnects Symposium, August 1995. Available from `http://www.hpl.hp.com/personal/John_Wilkes/papers/HamlynHotIntIII.pdf`.

[5] Cray Research, Inc., *Cray T3D System Architecture Overview*, March 1993.

[6] *Fiber-distributed data interface (FDDI)—Token ring media access control (MAC).* American National Standard for Information Systems ANSI X3.139-1987, July 1987. American National Standards Institute.

[7] L. Goldberg, *100Base-T4 transceiver simplifies adapter, repeater, and switch designs*, Electronic Design, (1995), pp. 155–160.

[8] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, *A high-performance, portable implementation of the MPI message passing interface standard.* Available from `http://www.mcs.anl.gov/mpi/mpicharticle/paper.html` or `ftp://ftp.mcs.anl.gov/pub/mpi/mpicharticle.ps`.

[9] R. Gusella, *A measurement study of diskless workstation traffic on Ethernet*, IEEE Transactions on Communications, 38 (1990).

[10] R. Horst, *TNet: A reliable system area network*, IEEE Micro, (1995), pp. 37–45.

[11] IBM Corporation, *Scalable POWERparallel System*, 1995. Available from `http://ibm.tc.cornell.edu/ibm/pps/sp2/sp2.html`.

[12] Intel Corporation, *Paragon XP/S Product Overview*, 1991.

[13] V. Karamcheti and A. Chien, *Software overhead in messaging layers: Where does the time go?*, in Proceedings of the Sixth Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI), 1994. Available from `http://www-csag.cs.uiuc.edu/papers/asplos94.ps`.

[14] M. J. Litzkow, M. Livny, and M. W. Mutka, *Condor—a hunter of idle workstations*, in Proceedings of the 8th International Conference of Distributed Computing Systems, June 1988, pp. 104–111.

[15] N. R. McKenzie, K. Bolding, C. Ebeling, and L. Snyder, *Cranium: An interface for message passing on adaptive packet routing networks*, in Proceedings of the 1994 Parallel Computer Routing and Communication Workshop, May 1994. Available from `ftp://shrimp.cs.washington.edu/pub/chaos/docs/cranium-pcrcw.ps.Z`.

[16] Message Passing Interface Forum, *The MPI message passing interface standard*, tech. rep., University of Tennessee, Knoxville, April 1994. Available from `http://www.mcs.anl.gov/mpi/mpi-report.ps`.

[17] R. Metcalfe and D. Boggs, *Ethernet: Distributed packet-switching for local computer networks*, Communications of the Association for Computing Machinery, 19 (1976), pp. 395–404.

[18] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, *Global Arrays: A portable "shared-memory" programming model for distributed memory computers*, in Supercomputing '94, 1994. Available from `http://www.computer.org/conferen/sc94/neiploch.html`.

[19] S. L. Scott, *Synchronization and communication in the T3E multiprocessor*, in Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), Cambridge, Massachusetts, October 1996, pp. 26–36. Available from `http://reality.sgi.com/sls_craypark/Papers/asplos96.html`.

[20] Silicon Graphics, Inc., *Origin Servers: Technical Overview of the Origin Family*, 1996. Available from `http://www.sgi.com/Products/hardware/servers/technology/overview.html`.

[21] P. G. Sobalvarro and W. E. Weihl, *Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors*, in Proceedings of the Parallel Job Scheduling Workshop at IPPS '95, 1995. Available from `http://www.psg.lcs.mit.edu/~pgs/papers/jsw-for-springer.ps`. Also appears in Springer-Verlag Lecture Notes in Computer Science, Vol. 949.

[22] C. B. Stunkel and W. K. Fuchs, *An analysis of cache performance for a hypercube multicomputer*, IEEE Transactions on Parallel and Distributed Systems, 3 (1992), pp. 421–432.

[23] J. Suplick, *An analysis of load balancing technology*. Available from `http://platform.com/products/lsf.comp.ps.Z`, January 1994.

[24] Thinking Machines Corporation, *The Connection Machine CM-5 Technical Summary*, 245 First Street, Cambridge, MA 02154-1264, October 1991.

[25] T. von Eicken, A. Basu, V. Buch, and W. Vogels, *U-Net: A user-level network interface for parallel and distributed computing*, in Proceedings of the 15th ACM Symposium on Operating Systems Principles, December 1995. Available from `http://www.cs.cornell.edu/Info/Projects/ATM/sosp.ps`.

[26] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser, *Active Messages: a mechanism for integrated communication and computation*, in Proceedings of the International Symposium on Computer Architecture, 1992.

[27] X3T11 Technical Committee, Project 1119D, *Fibre Channel Physical and Signalling Interface–3 (FC-PH-3), Revision 8.3*, American National Standards Institute, January 1996. Working draft proposed American National Standard for Information Systems. Available from `http://www.network.com/~ftp/FC/PH/FCPH3_83.ps`.

[28] S. Zhou, J. Wang, X. Zheng, and P. Delisle, *Utopia: A load sharing facility for large, heterogeneous distributed computer systems*, Software—Practice and Experience, 23 (1993), pp. 1305–1336. Als appeared as Technical Report CSRI-257, April, 1992. Available from `http://platform.com/products/lsf.paper.ps.Z`.